Constructing AudioUnit Plugins on the Web using Csound

Edward Costello, Victor Lazzarini, Joseph Timoney
Sound and Digital Music Technology Group
Maynooth University
Ireland
edward.costello@mumail.ie, victor.lazzarini@nuim.ie, joseph.timoney@nuim.ie

ABSTRACT

This paper describes a web-based application which can be used to construct AudioUnit plugins. Using this application the audio DSP component of an AudioUnit plugin can be created using the Csound audio programming language, and the user interface (UI) composed using HTML5. This is made possible using the combination of a Csound binary compiled for Google's portable native client API (PNaCl) which runs Csound code inside of the web application, and the Csound AudioUnit API which allows Csound compiled code to run inside of native AudioUnit plugins. The application allows users to live-code an AudioUnit plugin using on-the-fly Csound code and HTML5 evaluation within the browser which allows rapid testing of the plugin audio DSP and UI components. A common Javascript interface allows the plugin UI to communicate with both the web-based PNaCl Csound and the native AudioUnit encapsulated Csound instances. This common interface uses a JSON representation of the plugin control parameters and presets allowing users to control the plugin and create presets that function inside of both the web application and the AudioUnit.

1. INTRODUCTION

Audio plugins have become and important tool for the production of computer music as they allow composers and audio engineers to extend the sonic capabilities of audio software. Creating audio plugins can however be a non-trivial process involving the design and implementation of audio DSP components and plugin UI within a C/C++ based software framework. We have developed the Csound AudioUnit Factory web application which allows users to more easily create custom audio plugins that use Apple's AudioUnit framework for OS X. This application can construct AudioUnit plugins using the Csound language and HTML5 to create the audio DSP and UI components.

Creating AudioUnits using Csound enables users to more easily implement their plugin audio DSP engine by using a specialised audio programming language. Csound comes with a large array of unit generators and instrument examples available making the process of designing and implementing a plugin audio engine less time-consuming than coding each component from scratch using C/C++.

There are a number of related projects which also allow developers to create audio plugins using web technologies. Audiounitjs[1] is an Xcode project scaffold which can be used to produce AudioUnit plugins or OS X and iOS audio applications that contain HTML5 UIs. In this case communication between the native audio DSP backend and the UI is handled using the Webkit API. Jari Kleimola and Oliver Larkin's Web Audio Modules (WAMs)[4] define a web native audio module standard comparable to audio plugins. WAMs are constructed using a combination of a plain Javascript or Emscripten[11] compiled C/C++ audio processing component with the UI implemented using HTML5.

There are also projects such as Rory Walsh's Cabbage[9] which allow developers to create audio plugins or standalone applications using the Csound language. In this case the audio DSP component is written using the Csound language with the UI defined using a custom markup language embedded within a Csound CSD file.

The advantage of enabling users to write the plugin UI using web technologies over the Cabbage approach is that it allows developers to create user interface components which can leverage some of the many high quality HTML5 UI libraries available. Any components created may also be reused with minimal modification within other web-based applications. However, also allowing plugin audio DSP code to be written using Csound allows developers to make use of the large array of opcodes and instruments available for the language in their plugin projects. This is the approach taken for constructing audio plugins using the Csound AudioUnit Factory web application.

1.1 Csound

Csound[7] is a sound and music computing system that enables users to programmatically describe audio computation using a domain-specific language. It was originally developed by Barry Vercoe as a C-language version of his MUSIC 11 program[1] . Since then, it underwent significant change, becoming a general-purpose environment for music programming. Csound includes a large library of unit generators (*opcodes*), which can be used to construct audio digital signal processing programs. Code can be supplied to Csound as a text file in plain or CSD formats, or as text strings via UDP messages. Csound also provides a software API which allows third party developers to create applications using the

[1] https://github.com/russellmcc/audiounitjs

Web Audio Conference WAC-2016, April 4–6, 2016, Atlanta, USA

Csound engine. The API allows a host application to compile and run Csound code, as well as control its performance via messages, MIDI, OSC, etc.

1.2 AudioUnits

AudioUnits are an audio plugin standard developed by Apple for use in OS X and later iOS[2]. AudioUnit plugins enable the extensibility of audio software allowing the incorporation of additional audio processing components into applications such as digital audio workstations (DAWs). AudioUnit plugins come in a number of different types such as effect plugins which are used to process audio input and Music Device plugins which synthesise audio when triggered by MIDI events.

The audio DSP processing state a plugin is manipulated by altering the values of any exposed parameter controls. These may be altered in a number of ways such as in a plugin UI, or by an automation track within a DAW. The values of plugin parameters may also be altered by using the preset facility which allows the user to easily store and retrieve the plugin processing state.

AudioUnits may be developed and used within a host application without a custom UI. If a custom UI is not created for a plugin, a UI will be programmatically created using the plugins parameter properties to construct the relevant types of UI elements. If a custom UI is required it may be constructed using Apple's Interface builder or by using a custom UI toolkit.

AudioUnit plugins are commonly developed using a combination of C/C++ and Objective-C Cocoa or by using a C++ based multi plugin development framework such as IPlug[3] or JUCE[4].

1.3 Csound Portable Native Client

Csound Portable Native Client (PNaCl) is an implementation of Csound that can run under supported browsers (e.g Chrome, Chromium). PNaCl is a part of Google's Native Client sandboxing technology that allows Chrome to safely run plugins on a web page[10][8]. In order to accomplish this PNaCl uses a subset of LLVM bytecode[2], which is translated to the host browser CPU architecture at runtime allowing compiled PNaCl executables to run natively across many platforms.

The Csound PNaCl API uses a Javascript interface to control the running Csound instance. This gives developers access to various aspects of the engine, including realtime Audio/MIDI input, access to control channels, and the ability to compile orchestra code while executing a performance.

2. CSOUND AUDIOUNIT API

The Csound AudioUnit API[5] is a software framework for developing AudioUnit plugins using the Csound language. Plugin UIs may also be developed using the Csound AudioUnit API using the provided Objective-C/Cocoa or HTML5 interface.

The audio DSP component of plugins developed using the API are written using the Csound language inside of a CSD-format file. Any of the plugin associated control parameters or presets are defined as JSON objects within the Parameters.json and Presets.json files respectively. All of the AudioUnit configuration properties are also defined as JSON objects inside of the plugin Configuration.json file. This file defines a number of properties which are read by the Csound AudioUnit API, some of which are related to locating resources and instantiating plugin UI.

2.1 Plugin DSP

As indicated above, the audio DSP component of a plugin is created using the Csound language. When constructing an effect plugin the audio IO is handled using a combination of the *ins* and *outs* opcodes. These opcodes ensure that audio data is received from the AudioUnit host and any processed audio is copied back to the host audio input channel.

When constructing an audio synthesiser plugin, instead of receiving an audio input, Csound receives MIDI data from the plugin host to trigger the necessary instruments within the CSD file. MIDI is received by an instrument using a combination of the *massign* opcode which assigns incoming MIDI from a specified channel to a named instrument, and the numerous MIDI data translation opcodes such as *cpsmidi* which translates from a MIDI note number to a frequency value in Hertz, and *ampmidi* which translates from a MIDI velocity value to an amplitude value scaled to Csound full-scale (*0dbfs*) value.

2.2 Plugin Parameters & Presets

Any audio DSP values which require user control such as the instrument gain, or a filter cutoff are defined as plugin parameters using a JSON file entitled Parameters.json, and are received inside of Csound instruments using the *chnget* opcode. The Parameters.json file defines a JSON array that contains a number of parameter objects. Each parameter object contains a number of properties, some of which are mandatory for a parameter to be registered correctly. The mandatory properties fields are:

- **name** the parameter name identifier which is used by the *chnget* Csound opcode
- **minValue** the minimum numeric value of the parameter
- **maxValue** the maximum numeric value of the parameter
- **defaultValue** the default numeric value of the parameter

The corresponding values of optional unit and flag parameter properties are declared inside of the AudioUnit API AudioUnitProperties.h file.

These optional parameter properties are:

- **unit** indicates the unit type of a parameter such as Hertz, decibels or indices
- **flag** which may set a number of parameter value options such as how the parameter should be displayed e.g. logarithmically, exponentially
- **strings** an array of strings useful inside of the default AudioUnit UI for when the parameter unit value is

[2]https://developer.apple.com/library/mac/documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide
[3]https://github.com/olilarkin/wdl-ol
[4]http://www.juce.com/
[5]https://github.com/eddyc/CsoundAU

set to indices, each string in the array is used to represent the corresponding index value within a drop-down menu widget

A simple audio effect with a single gain parameter could be constructed using the Csound orchestra code in code example 1 and the Parameters.json file in code example 2:

Code Example 1: *Csound Instrument*

```
instr Gain

    aInputL, aInputR ins
    kGain chnget "Gain"
    aOutputL = aInputL * kGain
    aOutputR = aInputR * kGain
    outs aOutputL, aOutputR

endin
```

Code Example 2: *Parameters.json*

```
[
    {
        "name":"Gain",
        "minValue": 0,
        "maxValue": 1,
        "defaultValue":0.75
    }
]
```

Presets may also be added to an AudioUnit by listing them within the Presets.json file. Each preset is defined as an object within a Javascript array that contains two properties:

- **name** the preset name as a string
- **preset** an object containing name value pairs that correspond to each parameter name and preset value

If a parameter is defined in Parameters.json but not specified within a preset, the parameter default value is used. Code example 3 shows an example preset for the previous instrument and parameter example setting the *Gain* value parameter to 0.5.

Code Example 3: *Presets.json*

```
[
    {
        "name":"Half Gain",
        "parameters": {
                            "Gain":0.5
                      }
    }
]
```

2.3 Plugin UI

The Csound AudioUnit API allows developers to create AudioUnit UIs using either Objective-C/Cocoa or using HTML5. As the HTML5 UI is used inside of plug-ins produced by Csound AudioUnit Factory only this facility will be explained in detail here.

AudioUnit UIs are commonly instantiated using a Cocoa view and communicate with the running AudioUnit instance using the AUEvent and AUParameter functions. Similar to the approach taken by audiounitjs, when using an HTML5 view with Csound AudioUnit API a Webkit Webview is instantiated within the plugin Cocoa view which allows the developer to define the plugin UI using HTML5 and to communicate with the running AudioUnit instance through Javascript. The pixel dimensions and the main HTML file name of the UI are specified inside of the Configuration.json file along with the name of the Csound CSD file. For example, if a plugin UI is required with dimensions 300 pixels wide and 400 pixels high with the main HTML file entitled index.html, we would write a Configuration.json file as shown in code example 4:

Code Example 4: *Configuration.json*

```
{
    "ViewBundleID":"AudioUnitView",
    "ViewType":"HTML",
    "ViewFileName":"index",
    "ViewWidth":"300",
    "ViewHeight":"400",
    "csd":"main"
}
```

Using the Webkit API an AudioUnit Javascript object is instantiated before the main HTML file has loaded which implements two methods that facilitate communication between the Webview and the AudioUnit:

- **setParameter(name, value)**
  This method receives as its first argument the name of an AudioUnit parameter as defined within Parameters.json and sets the corresponding parameter to the value given by the second argument.
- **getParameterCallback(name, callback)**
  This method receives as its first argument the name of an AudioUnit parameter as defined in the Parameters.json file and a callback function as its second argument. The callback function uses the signature: **callback(value)**. When the corresponding parameter has been altered such as when a preset is selected, the callback function is called with the corresponding preset value.

Code example 5 shows how the AudioUnit Javascript object may be used to control an AudioUnit parameter using a slider UI element.

Code Example 5: *index.html*

```
<html>
    <body>
    <input type="range"
           id="Gain"
           min="0"
           max="1"
           step="0.001">
    Gain
    </input>
    <script>

        var slider = document.getElementById("
            Gain");
        slider.oninput = function() {

            AudioUnit.setParameter("Gain",
                           slider.value);
        };
```

```
        AudioUnit.getParameterCallback("Gain",
            function(gainValue) {

                slider.value = gainValue;
            }
        );
    </script>
</body>
</html>
```

3. CSOUND AUDIOUNIT FACTORY

The Csound AudioUnit Factory is a web application that allows users to live-code an AudioUnit plugin using the Csound language and HTML5 which can then be used within an AudioUnit host such as Ableton Live or Logic.

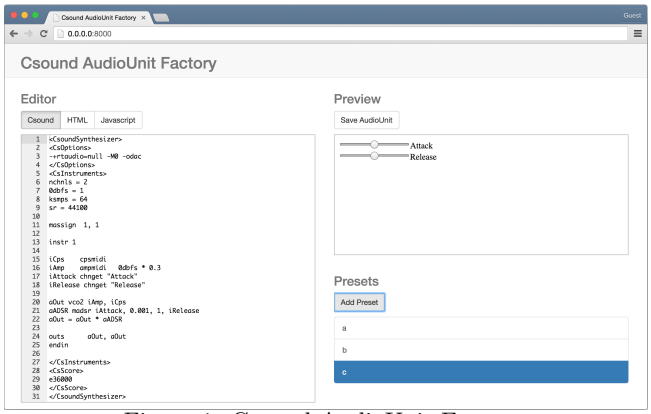

Figure 1: Csound AudioUnit Factory

3.1 Application Interface

The application UI consists of a text editor component, a preview iframe and a modal dialogue for creating presets. The text editor component is made up of three ace editor[6] instances which can be selected and displayed individually using a button which corresponds to the editor mode. The editor has three modes which are used to edit the plugin Csound CSD file, the UI main HTML file and also its main Javascript file. When the application has loaded, a code template for each mode is read from the server and inserted into the appropriate ace instance. The user may evaluate the contents of the text editor in any mode which will update either the running Csound instance or the iframe preview.

Any widgets within the preview iframe can also control the state of the running Csound instance when connected using the appropriate Csound opcodes and Javascript methods.

3.2 Audio DSP Component

This application uses the PNaCl version of Csound for the audio DSP backend. Although there is also a Javascript Csound library available which is built using Emscripten, due to performance considerations the PNaCl library was used within this project.

When the application has loaded and Csound has been instantiated, a template Csound CSD file is immediately compiled and performed. When the user evaluates Csound code within the text editor the code string is sent to the Csound instance and evaluated using the CsoundCompileOrc method. Once the code evaluation has been executed any changes to the Csound code will be present when the relevant instruments are next instantiated.

3.3 UI Preview Frame

The UI preview frame is an iframe which dynamically renders the contents of the HTML and Javascript editors when the user evaluates code within either editor. In order to facilitate communication between a HTML5 based UI inside of either a running AudioUnit or the AudioUnit Factory application, an AudioUnit Javascript object is injected into the running iframe instance before the user Javascript code is evaluated. This AudioUnit object implements the same method signatures as those used within the Webkit view in the AudioUnit plugin UI. When the **setParameter** method is called, a message is sent to the iframe parent window which calls the PNaCl Csound SetChannel method with the named channel string and corresponding value. In the case of the **getParameterCallback** method, the specified callback is added to an object within the iframe scope which stores each callback as a value for the corresponding control property. Callbacks are executed when a message has been received from the iframe host window with the corresponding control name string and value. This method is also used to register the AudioUnit parameters instead of using a Parameters.json file. If for example an *input* element is registered, the input element id, minimum, maximum and current value are stored as the AudioUnit parameter name, minimum, maximum and default values.

3.4 Presets

Presets are added to the AudioUnit using the *Add Presets* button in the application interface. When this button is pressed, a modal view appears where the user can name the preset and, clicking the modal view *Add* button, can add the preset to the preset selection list beneath the preview iframe. When a preset is added, the value property of each of the registered parameters is read and stored within a Javascript array. When a preset within the preset list is selected, each of the parameter values are read from the preset array and the appropriate Csound input channel and HTML elements are set to the corresponding values.

3.5 Saving AudioUnits

In order to construct an AudioUnit plugin, the Csound AudioUnit Factory contains an AudioUnit plugin pre-compiled using the Csound AudioUnit API. As plugins compiled using this API have their audio DSP engine and UI defined using Csound and HTML5 code, creating a plugin is a matter of editing these resources within the AudioUnit plugin. The relevant resources which need to be modified are: the Parameters.json and Presets.json files which contain the plugin parameter controls and presets, the main Csound CSD file, and for the UI the main HTML and Javascript files.

The AudioUnit plugin is built using the JSZip[7] library. An archive of the pre-compiled AudioUnit is opened inside of the web application using JSZip and each of the required files are overwritten within the archive using JSZip **file** method which takes as its arguments the path where the file will be written and a string representing the files content.

[6] http://ace.c9.io/

[7] https://stuk.github.io/jszip/

The Csound, HTML and Javascript file strings are read from their associated ace editor instances while the Parameters.json and Presets.json file content strings are created using the application parameters and presets arrays converted to strings using the JSON.stringify method.

When an AudioUnit has been constructed it can be saved and used within a plugin host by clicking the *Save AudioUnit* button which will execute the appropriate functions for reading the necessary file data and adding it to the pre-compiled AudioUnit zip file. When this process is completed the AudioUnit zip file is saved to the client downloads folder. The AudioUnit may then be used within a host application by unzipping the plugin and placing it within one of OS X AudioUnit search directories.

4. LIMITATIONS & FUTURE WORK

As this project is in its early stages there are still a number of limitations and areas which may be improved with further development. This application uses Google's PNaCl runtime for the audio engine which makes it currently only possible to run the application under browsers that support this plugin model (e.g. Chrome, Chromium). As mentioned previously, there is also an Emscripten compiled version of the Csound library. This was tested and deemed unsuitable for this project due to performance issues. It is hoped that these issues may be resolved in the future and the project can support many more browsers using the pure Javascript Csound library.

It would also be desirable to implement a number of common features found within integrated development environments which may further simplify the plugin development process. Features such as project templates/examples, code completion, and Csound syntax highlighting would make useful additions to the project. The current design of the text editing area of the project is also limiting as it does not allow the use of any source files additional to the provided HTML, Javascript and Csound files. Allowing users to freely create source files for their projects or to import libraries in a similar manner to jsfiddle[8] would allow users to create more advanced audio DSP engines and UIs within the application.

At present this project is also limited to creating AudioUnit Music devices which are used for constructing synthesiser style plugins. In order to create effects plugins an interface for testing plugin effects using audio file or line input must be implemented. In the future is is also envisioned that plugins could be created for a number of platforms and using standards other than AudioUnits such as VST[9] or LV2[10].

5. CONCLUSIONS

In this paper we have presented a web application for creating AudioUnit plugins using the Csound language and HTML5. Writing AudioUnit plugins using Csound and HTML5 enables developers to more easily create and reuse plugin components such as Csound instruments and UI controls within other applications.

This application framework also allows for the future expansion of possible output plugin formats compiled for different system architectures enabling plugin developers to more easily design and implement audio plugins that work within a large number audio production software suites.

Csound AudioUnit Factory can be used online at http://eddyc.github.io/Csound-AudioUnit-Factory/.

[8]https://jsfiddle.net/

[9]http://www.steinberg.net/en/company/technologies.html

[10]http://lv2plug.in/

6. ACKNOWLEDGMENTS

This research was partly funded by the Program of Research in Third Level Institutions (PRTLI 5) of the Higher Education Authority (HEA) of Ireland, through the Digital Arts and Humanities programme.

7. REFERENCES

[1] R. Boulanger. *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. MIT press, 2000.

[2] A. Donovan, R. Muth, B. Chen, and D. Sehr. PNaCl: Portable Native Client Executables. *Google White Paper*, 2010.

[3] J. Kleimola. Daw plugins for web browsers. 1st Web Audio Conference, IRCAM, Paris, 2015.

[4] J. Kleimola and O. Larkin. Web audio modules. SMC, Maynooth University, 2015.

[5] V. Lazzarini, E. Costello, S. Yi, and J. ffitch. Csound on the Web. In *Linux Audio Conference*, pages 77–84, Karlsruhe, Germany, May 2014.

[6] V. Lazzarini, E. Costello, S. Yi, and J. ffitch. Extending csound to the web. 1st Web Audio Conference, IRCAM, Paris, 2015.

[7] V. Lazzarini, J. ffitch, S. Yi, O. Brandtsegg, J. Heintz, and I. McCurdy. *Csound: A Sound and Music Programming System*. Springer, 2016.

[8] D. Sehr, R. Muth, C. BifiňĆe, V. Khimenko, E. Pasko, K. Schimpf, B. Yee, and B. Chen. Adapting Software Fault Isolation to Contemporary CPU Architectures. In *19th USENIX Security Symposium*, 2010.

[9] R. Walsh. Audio plugin development with cabbage. In *Proceedings of the Linux Audio Conference, Maynooth, Ireland*, pages 47–53, 2011.

[10] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *2009 IEEE Symposium on Security and Privacy*, 2009.

[11] A. Zakai. Emscripten: an LLVM-to-Javascript Compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications*, pages 301–312. ACM, 2011.